更新时间：2021年6月26日 @亮子力

源码目录：

主目录\foundation\communication\softbus_lite

```
├── authmanager【提供设备认证机制和设备知识库管理】
├── discovery【提供基于coap协议的设备发现机制】
├── interfaces
├── os_adapter【操作系统适配层】
├── trans_service【提供认证和数据传输通道】
└── BUILD.gn
通过BUILD.gn的分析，我们知道整个softbus_lite目录下的所有源码文件将被编译到一个动态库中。
其他依赖软总线的模块在编译的时候加上这个动态库的依赖就可以了。
例如：分布式调度子系统所在的foundation这个bin文件的编译就依赖这个动态库。
```

authmanager【提供设备认证机制和设备知识库管理】

```
当发现有请求时，调用ProcessDataEvent函数，收包，检验包头，根据数据包的类型确定不同的处理方式。
类型主要包括以下三种：
MODULE_AUTH_SDK       加密数据类型
MODULE_TRUST_ENGINE 可信类型，直接进行数据传输
MODULE_CONNECTION   进行ip及设备认证

├── BUILD.gn
├── include
│   ├── auth_conn.h
│   ├── auth_interface.h
│   ├── bus_manager.h
│   ├── msg_get_deviceid.h
│   └── wifi_auth_manager.h
└── source
    ├── auth_conn.c【提供发送、接收、认证、获取秘钥功能】
    ├── auth_interface.c【管理各个会话节点、各个链接节点、各个秘钥节点，提供包括增删改查等功
能】
    ├── bus_manager.c【主要通过deviceIp创建两个不同的listen，主要用来监听系统上有哪些
device及新的device节点的创建;其中有两个回调函数OnConnectEvent和OnDataEvent,分别是用来处理
设备节点的基本操作及节点数据的处理】
    ├── msg_get_deviceid.c【提供以cJSON格式获取各个设备的信息，包括设备id、链接信息、设备
名、设备类型等】
    └── wifi_auth_manager.c【主要实现了连接管理和数据接收功能。连接管理包括连接的建立、断开
及连接的查找。数据接收包括数据获取、包头及包长等的校验，并且为了简化包头数据读取，单独实现了对一
个int型和一个long型数据的接收函数】
```

discovery【提供基于coap协议的设备发现机制】

```
├── BUILD.gn
├── coap【目录主要是负责COAP协议的部分】
│   ├── include
│   │   ├── coap_adapter.h
│   │   ├── coap_def.h
│   │   ├── coap_discover.h
│   │   ├── coap_socket.h
│   │   ├── json_payload.h
│   │   ├── nstackx_common.h
│   │   ├── nstackx_database.h
│   │   ├── nstackx_device.h
│   │   ├── nstackx_error.h
│   │   └── nstackx.h
│   └── source
│       ├── coap_adapter.c
│       ├── coap_discover.c【实现了基于COAP的设备发现功能】
│       ├── coap_socket.c
│       ├── json_payload.c
│       ├── nstackx_common.c
│       └── nstackx_device.c
└── discovery_service【基于coap协议实现了轻量设备端的服务发布的能力】
    ├── include
    │   ├── coap_service.h
    │   ├── common_info_manager.h
    │   └── discovery_error.h
    └── source
        ├── coap_service.c
        ├── common_info_manager.c
        └── discovery_service.c
```

```
discovery的实现前提是确保发现端设备与接收端设备在同一个局域网内且能互相收到对方的报文。大致流程是：

发现端设备，使用coap协议在局域网内发送广播；
接收端设备使用PublishService接口发布服务，接收端收到广播后，发送coap协议单播给发现端；
发现端设备收到报文会更新设备信息。
```

os_adapter【操作系统适配层】

```
├── include
│   └── os_adapter.h
└── source
    ├── L0
    │   └── os_adapter.c
    └── L1
        └── os_adapter.c
```

trans_service【提供认证和数据传输通道】

```
它主要封装了socket、cJSON、线程锁接口，实现了用户的创建、监听、会话管理，以及设备、指令、数据等信息的获取，最终提供加密和解密传输两种传输通道。
├── BUILD.gn
├── include
│   ├── libdistbus
│   │   ├── auth_conn_manager.h
│   │   └── tcp_session_manager.h
│   └── utils
│       ├── aes_gcm.h
│       ├── comm_defs.h
│       ├── data_bus_error.h
│       ├── message.h
│       └── tcp_socket.h
└── source
    ├── libdistbus
    │   ├── auth_conn_manager.c      【用户创建，监听，连接等服务管理】
    │   ├── tcp_session.c            【会话管理】
    │   ├── tcp_session.h
    │   ├── tcp_session_manager.c
    │   ├── trans_lock.c             【互斥锁初始化以及互斥锁资源获取与释放】
    │   └── trans_lock.h
    └── utils
        ├── aes_gcm.c        【提供加密传输和解密传输接口】
        ├── message.c        【用于获取以cJSON格式管理的设备（包括设备名、设备类型、设备ID等)、指令、数据、会话（包括用户端口、会话端口等)等信息】
        └── tcp_socket.c     【端口号管理以及数据传输管理】
```

一、如何初始化软总线

调用软总线模块并不复杂，需要准备2个结构体就可以召唤神龙，简称StartBus()。以下我们以Hi3861为例写一个简单的例子。

```
static void InitSoftbus(void)
{printf(">>>>>%s:%d:%s()\n",__FILE__,__LINE__,__FUNCTION__);
```

```
    static PublishInfo g_publishInfo = {
        .capabilityData = (unsigned char *)"1",
        .capability = "ddmpCapability",
        .dataLen = 1,
        .publishId = 1,
        .mode = DISCOVER_MODE_ACTIVE,
        .medium = COAP,
        .freq = MID,
    };
    static IPublishCallback g_publishCallback = {
        .onPublishSuccess = OnSuccess,
        .onPublishFail = OnFail,
    };

    int ret = PublishService(g_demoModuleName, &g_publishInfo,
&g_publishCallback);
    if (ret != 0) {
        printf("PublishService err\n");
    }
    sleep(5);
    ret = CreateSessionServer(g_demoModuleName, g_demoSessionName,
&g_sessionCallback);
    if (ret != 0) {
        printf("CreateSessionServer err\n");
    }
    printf("InitSoftbus ok\n");
}
```

一、被发现端，发布服务。【discovery目录】

在分布式软总线子系统中，设备分为发现端和被发现端。这里我们使用手机作为发现端，Hi3861开发板发布服务，作为被发现端。

发布服务主要是通过PublishService()这个函数，相关的代码在：主目录\foundation\communication\softbus_lite\discovery_service文件夹下。

如果设备要发布软总线服务，就需要调用下面这个函数。

```
//moduleName：调用模块名
//info：PublishInfo结构体
//cb：发布成功或者失败的回调函数
int PublishService(const char *moduleName, const struct PublishInfo *info, const
struct IPublishCallback *cb)
{printf(">>>>>%s:%d:%s()[soft bus init
start]\n",__FILE__,__LINE__,__FUNCTION__);
    //许可检查，首先将检查是否有发布权限。这里调用的os_adapter【操作系统适配层】相关部分。
    if (SoftBusCheckPermission(SOFTBUS_PERMISSION) != 0 || info == NULL || cb ==
NULL) {
        SOFTBUS_PRINT("[DISCOVERY] PublishService invalid para(info or cb)\n");
        return ERROR_INVALID;
    }

    //参数有效性检查，检查了一些发布参数的合法性，其次确认发布协议是不是COAP。
    if (moduleName == NULL || strlen(moduleName) >= MAX_PACKAGE_NAME || info-
>publishId <= 0 ||
        info->dataLen > MAX_CAPABILITY_DATA_LEN) {
        SOFTBUS_PRINT("[DISCOVERY] PublishService invliad para\n");
        PublishCallback(info->publishId, PUBLISH_FAIL_REASON_PARAMETER_INVALID,
NULL, cb);
        return ERROR_INVALID;
    }//如果参数检查失败，调用PublishCallback()来回调cb里面的失败回调函数
    if (info->medium != COAP) {
        PublishCallback(info->publishId, PUBLISH_FAIL_REASON_NOT_SUPPORT_MEDIUM,
NULL, cb);
        return ERROR_INVALID;
    }

    //用g_discoveryMutex防止多个设备对外发布服务产生的冲突
    if (g_discoveryMutex == NULL) {
        g_discoveryMutex = MutexInit();
        if (g_discoveryMutex == NULL) {
            PublishCallback(info->publishId, PUBLISH_FAIL_REASON_UNKNOWN, NULL,
cb);
            return ERROR_FAIL;
        }
    }
    MutexLock(g_discoveryMutex);//解除互斥锁

    //InitService()这个函数比较重要。参考1.初始化InitService()发现服务
    if (InitService() != ERROR_SUCCESS) {
        SOFTBUS_PRINT("[DISCOVERY] PublishService InitService fail\n");
        PublishCallback(info->publishId, PUBLISH_FAIL_REASON_UNKNOWN, NULL, cb);
        MutexUnlock(g_discoveryMutex);
        return ERROR_FAIL;
    }

    //AddPublishModule()函数，参考2.模块增加到g_publishModule
    PublishModule *findModule = AddPublishModule(moduleName, info);
    if (findModule == NULL) {
        SOFTBUS_PRINT("[DISCOVERY] PublishService AddPublishModule fail\n");
        PublishCallback(info->publishId, PUBLISH_FAIL_REASON_UNKNOWN, NULL, cb);
        MutexUnlock(g_discoveryMutex);
        return ERROR_FAIL;
    }

    //注册COAP服务，参考3.CoapRegisterDefualtService()
    int ret = ERROR_SUCCESS;
    if (info->capability == NULL || info->capabilityData == NULL) {
        (void)CoapRegisterDefualtService();
    } else {
        ret = DoRegistService(info->medium);
    }

    MutexUnlock(g_discoveryMutex);

    //这个PublishCallback()就很简单了，上面出现好多次了，初始化成功或者失败的回调函数。
    if (ret != ERROR_SUCCESS) {
        PublishCallback(info->publishId, PUBLISH_FAIL_REASON_UNKNOWN,
findModule, cb);
        return ERROR_FAIL;
    } else {
        PublishCallback(info->publishId, ERROR_SUCCESS, findModule, cb);
        return ERROR_SUCCESS;
    }
}
```

1.初始化InitService()发现服务

```
Z:\harmony110\foundation\communication\softbus_lite\discovery\discovery_service\source\discovery_service.c
int InitService(void)
{
    if (g_isServiceInit != 0) {//全局变量g_isServiceInit：确认服务是否启动。如果其他模块启动过，那么直接返回成功。
        return ERROR_SUCCESS;
    }

    //初始化g_deviceInfo结构体。参考1.1.初始化g_deviceInfo结构体，相关文件【common_info_manager.c】
    if (InitCommonManager() != 0) {
        SOFTBUS_PRINT("[DISCOVERY] InitService InitCommonManager fail\n");
        DeinitService();
        return ERROR_FAIL;
    }

    //分配内存，参考：1.2.初始化全局g_publishModule和g_capabilityData
    g_publishModule = calloc(1, sizeof(PublishModule) * MAX_MODULE_COUNT);
    if (g_publishModule == NULL) {
        DeinitService();
        return ERROR_NOMEMORY;
    }
    g_capabilityData = calloc(1, MAX_SERVICE_DATA_LEN);
    if (g_capabilityData == NULL) {
        DeinitService();
        return ERROR_NOMEMORY;
    }
    //软总线注册第一个监听函数WifiEventTrigger()，当接入网络时trigger
    //注册wifi Callback，将WifiEventTrigger()↓函数，赋值给全局变量g_wifiCallback，参考b.COAP初始化wifi事件
    RegisterWifiCallback(WifiEventTrigger);

    //COAP初始化。参考1.3.初始化COAP协议服务，相关文件【coap_service.c】
    int ret = CoapInit();
    if (ret != ERROR_SUCCESS) {
        SOFTBUS_PRINT("[DISCOVERY] InitService CoapInit fail\n");
        DeinitService();
        return ret;
    }printf(">>>>>%s:%d[Register WifiEventTring()]\n",__FILE__,__LINE__);//添加打印调试

    //COAP写入消息队列，这样会触发上面↑消息回调函数，WifiEventTrigger()，这个函数比较重要，TODO 1.4.
    CoapWriteMsgQueue(UPDATE_IP_EVENT);

    //COAP注册设备信息
    ret = CoapRegisterDeviceInfo();
    if (ret != ERROR_SUCCESS) {
        SOFTBUS_PRINT("[DISCOVERY] InitService CoapRegisterDeviceInfo fail\n");
        DeinitService();
        return ret;
    }

    g_isServiceInit = 1;
    SOFTBUS_PRINT("[DISCOVERY] InitService ok\n");
    return ERROR_SUCCESS;
}
```

1.1.初始化g_deviceInfo结构体

```
Z:\harmony110\foundation\communication\softbus_lite\discovery\discovery_service\
source\common_info_manager.c

int InitCommonManager(void)
{
    if (InitLocalDeviceInfo() != 0) { //初始化本地设备信息
        SOFTBUS_PRINT("[DISCOVERY] InitCommonManager fail\n");
        return ERROR_FAIL;
    }
    return ERROR_SUCCESS;
}
================================================================================
================
int InitLocalDeviceInfo(void) //初始化本地设备信息
{
    char deviceId[DEVICEID_MAX_NUM] = {0};

    if (g_deviceInfo != NULL) { //初始化一个g_deviceInfo的结构体。包含本地设备的各种信
息。
        memset_s(g_deviceInfo, sizeof(DeviceInfo), 0, sizeof(DeviceInfo));
    } else {
        g_deviceInfo = (DeviceInfo *)calloc(1, sizeof(DeviceInfo));
        if (g_deviceInfo == NULL) {
            return ERROR_FAIL;
        }
    }

    g_deviceInfo->devicePort = -1;      //默认，g_deviceInfo.设备端口-1
    g_deviceInfo->isAccountTrusted = 1; //默认，g_deviceInfo.账户是可信的

    unsigned int ret;
    //从文件获取设备id，这个函数还有好几层，就不继续深挖了。大体就是从DEVICE_ID_FILE文件中读
取。
    //#define DEVICE_ID_FILE   "/storage/data/softbus/deviceid"
    ret = GetDeviceIdFromFile(deviceId, MAX_VALUE_SIZE);
    if (ret != ERROR_SUCCESS) {
        SOFTBUS_PRINT("[DISCOVERY] Get device fail\n");
        return ERROR_FAIL;
    }

#if defined(__LITEOS_M__) || defined(__LITEOS_RISCV__) //g_deviceInfo.设备的类型，
目前分为L0和L1
    g_deviceInfo->deviceType = L0;
    ret = (unsigned int)strcpy_s(g_deviceInfo->deviceName, sizeof(g_deviceInfo-
>deviceName), L0_DEVICE_NAME);
#else
    g_deviceInfo->deviceType = L1;
    ret = (unsigned int)strcpy_s(g_deviceInfo->deviceName, sizeof(g_deviceInfo-
>deviceName), L1_DEVICE_NAME);
#endif

    ret |= (unsigned int)strcpy_s(g_deviceInfo->deviceId, sizeof(g_deviceInfo-
>deviceId), deviceId);//g_deviceInfo.设备id
    ret |= (unsigned int)strcpy_s(g_deviceInfo->version, sizeof(g_deviceInfo-
>version), "1.0.0");//g_deviceInfo.版本号
    if (ret != 0) {
        return ERROR_FAIL;
    }

    SOFTBUS_PRINT("[DISCOVERY] InitLocalDeviceInfo ok\n");
    return ERROR_SUCCESS;
}
```

1.2.初始化全局g_publishModule和g_capabilityData

```
Z:\harmony110\foundation\communication\softbus_lite\discovery\discovery_service\
source\discovery_service.c
//首先，discovery_service.c开头定义了这两个全局变量。

PublishModule *g_publishModule = NULL;   //存放发布的模块
char *g_capabilityData = NULL;           //能力描述数据

typedef struct {
    char package[MAX_PACKAGE_NAME];
    int publishId;
    unsigned short medium;
    unsigned short capabilityBitmap;
    char *capabilityData;
    unsigned short dataLength;
    unsigned short used;
} PublishModule;//发布模块结构体

//初始化InitService()发现服务后，会将模块注册到全局变量g_publishModule中，服务注册到
g_capabilityData中。
```

1.3.初始化COAP协议服务

```
Z:\harmony110\foundation\communication\softbus_lite\discovery\discovery_service\
source\coap_service.c
int CoapInit(void)
{
    int ret = NSTACKX_Init();
    if (ret != 0) {
        SOFTBUS_PRINT("[DISCOVERY] CoapInit NSTACKX_Init fail\n");
        return ERROR_FAIL;
    }
    return ERROR_SUCCESS;
}
================================================================================
================
Z:\harmony110\foundation\communication\softbus_lite\discovery\coap\source\nstack
x_common.c
int NSTACKX_Init()
```

```
{
    int ret;
    if (g_nstackInitState != NSTACKX_INIT_STATE_START) {
        return NSTACKX_EOK;
    }

    g_nstackInitState = NSTACKX_INIT_STATE_ONGOING;
    cJSON_InitHooks(NULL);

    ret = CoapInitDiscovery(); //启动COAP端口监听
    if (ret != NSTACKX_EOK) {
        goto L_ERR_INIT;
    }
    g_nstackInitState = NSTACKX_INIT_STATE_DONE;
    return NSTACKX_EOK;

L_ERR_INIT:
    ret = NSTACKX_Deinit();
    if (ret != NSTACKX_EOK) {
        SOFTBUS_PRINT("[DISCOVERY] deinit fail\n");
    }
    return NSTACKX_EFAILED;
}
================================================================================
================
Z:\harmony110\foundation\communication\softbus_lite\discovery\coap\source\coap_d
iscover.c
int CoapInitDiscovery(void)
{
    int ret = CoapInitSocket(); //参考a.COAP初始化Socket //启动监听端口
    if (ret != NSTACKX_EOK) {
        SOFTBUS_PRINT("[DISCOVERY] Init socket fail\n");
        return ret;
    }

    ret = CoapInitWifiEvent();  //参考b.COAP初始化wifi事件
    if (ret != NSTACKX_EOK) {
        SOFTBUS_PRINT("[DISCOVERY] Init wifi event fail\n");
        return ret;
    }
#if defined(__LITEOS_A__)
    ret = CreateQueryIpThread();
    if (ret != NSTACKX_EOK) {
        SOFTBUS_PRINT("[DISCOVERY] Init query Ip fail\n");
        return ret;
    }
#endif
    if (CreateMsgQueThread() != NSTACKX_EOK) {
        return NSTACKX_EFAILED;
    }

    return CreateCoapListenThread();    //参考c.创建COAP监听线程
}
================================================================================
================
```

a.COAP初始化Socket

```
//这个函数跳转到了coap【目录主要是负责COAP协议的部分】
Z:\harmony110\foundation\communication\softbus_lite\discovery\coap\source\coap_s
ocket.c
int CoapInitSocket(void)
{
    if (g_serverFd >= 0) {
        return NSTACKX_EOK;
    }
    struct sockaddr_in sockAddr;
    (void)memset_s(&sockAddr, sizeof(sockAddr), 0, sizeof(sockAddr));
    sockAddr.sin_port = htons(COAP_DEFAULT_PORT);
    g_serverFd = CoapCreateUdpServer(&sockAddr);    //创建一个udp服务器，这个socket
赋值给g_serverFd
    if (g_serverFd < 0) {
        return NSTACKX_OVERFLOW;
    }
    COAP_SoftBusInitMsgId();
    return NSTACKX_EOK;
}
================================================================================
================
int CoapCreateUdpServer(const struct sockaddr_in *sockAddr) //创建一个udp服务器
{
    if (sockAddr == NULL) {
        return NSTACKX_EINVAL;
    }

    struct sockaddr_in localAddr;
    socklen_t len = sizeof(localAddr);
    int sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd < 0) {
        return NSTACKX_OVERFLOW;
    }

    (void)memset_s(&localAddr, sizeof(localAddr), 0, sizeof(localAddr));
    localAddr.sin_family = AF_INET;             //结构体sockaddr_in.sin_family = 2
    localAddr.sin_port = sockAddr->sin_port;    //结构体sockaddr_in.sin_port =
5684，定义在coap_socket.h
    if (sockAddr->sin_addr.s_addr != 0) {
        localAddr.sin_addr.s_addr = sockAddr->sin_addr.s_addr;  //测试发现这里执行的
是else
    } else {
        localAddr.sin_addr.s_addr = htonl(INADDR_ANY);  //结构体
sockaddr_in.sin_addr.s_addr = 0 打印测试结果是0
    }

    if (bind(sockfd, (struct sockaddr *)&localAddr, len) == -1) {   //创建一个
socket，然后用bind()绑定到指定的IP+PORT
        CloseSocket(&sockfd);
        return NSTACKX_EFAILED;
    }

    if (getsockname(sockfd, (struct sockaddr *)&localAddr, &len) == -1) {   //获
取套接字名称
        CloseSocket(&sockfd);
        return NSTACKX_EFAILED;
    }
    return sockfd;
}
```

b.COAP初始化wifi事件

```
//基本原理就是，像wifi_lite注册事件回调函数
//最终调用调用WifiEventTrigger()函数
Z:\harmony110\foundation\communication\softbus_lite\discovery\coap\source\coap_d
iscover.c
int CoapInitWifiEvent(void)
{
    SOFTBUS_PRINT("[DISCOVERY] CoapInitWifiEvent\n");
    unsigned int ret;
    if (g_wifiQueueId == -1) {
        ret = CreateMsgQue("/wifiQue",      //创建一个消息队列
            WIFI_QUEUE_SIZE, (unsigned int*)&g_wifiQueueId,
            0, sizeof(AddressEventHandler));
        if (ret != 0) {
            SOFTBUS_PRINT("[DISCOVERY]CreateMsgQue fail\n");
            (void)CoapDeinitWifiEvent();
            return ret;
        }

#if defined(__LITEOS_M__) || defined(__LITEOS_RISCV__)
        g_coapEventHandler.OnWifiConnectionChanged =
CoapConnectionChangedHandler;//参考下面解析
        WifiErrorCode error = RegisterWifiEvent(&g_coapEventHandler);   //注册
wifi事件，全局变量g_coapEventHandler
        if (error != WIFI_SUCCESS) {
            SOFTBUS_PRINT("[DISCOVERY]RegisterWifiEvent fail, error:%d\n",
error);
            (void)CoapDeinitWifiEvent();
            g_wifiQueueId = -1;
            return error;
        }
#endif
    }
    return NSTACKX_EOK;
}
================================================================================
================
#if defined(__LITEOS_M__) || defined(__LITEOS_RISCV__)
static void CoapConnectionChangedHandler(int state, WifiLinkedInfo* info)
{
    (void)info;
    CoapWriteMsgQueue(state);//参考下面解析
}
================================================================================
================
void CoapWriteMsgQueue(int state)
{
    SOFTBUS_PRINT("[DISCOVERY] CoapWriteMsgQueue\n");
    AddressEventHandler handler;
    handler.handler = CoapHandleWifiEvent;//参考下面解析
    handler.state = state;
    /* while a new event coming, it must stop the previous loop 当一个新的事件出现
时，它必须停止之前的循环 */
    g_queryIpFlag = 0;
    (void)WriteMsgQue(g_wifiQueueId, &handler, sizeof(AddressEventHandler));
}
================================================================================
================
void CoapHandleWifiEvent(unsigned int para)
{
    if (g_wifiCallback != NULL) {   //g_wifiCallback在int InitService(void)
{RegisterWifiCallback(WifiEventTrigger);}
        g_wifiCallback(para);       //WifiEventTrigger()
    }
}
```

c.创建COAP监听线程, HandleReadEvent()

```
//创建了CoapReadHandle线程，处理COAP_DEFAULT_PORT端口上的UDP socket的数据（也就是基于
COAP协议的discover广播消息）
Z:\harmony110\foundation\communication\softbus_lite\discovery\coap\source\coap_d
iscover.c
int CreateCoapListenThread(void)
{
    g_terminalFlag = 1;

#if defined(__LITEOS_M__) || defined(__LITEOS_RISCV__)
    if (g_coapTaskId != NULL) {
        return NSTACKX_EOK;
    }

    osThreadAttr_t attr;            //创建一个osThreadAttr_t结构体
    attr.name = "coap_listen_task";
    attr.attr_bits = 0U;
    attr.cb_mem = NULL;
    attr.cb_size = 0U;
    attr.stack_mem = NULL;
    attr.stack_size = LOSCFG_BASE_CORE_TSK_DEFAULT_STACK_SIZE;
    attr.priority = osPriorityNormal4; // COAP_DEFAULT_PRIO -> cmsis prio
    //新建一个系统线程，全局变量g_coapTaskId
    g_coapTaskId = osThreadNew((osThreadFunc_t)CoapReadHandle, NULL, &attr);
//参考下面解析CoapReadHandle
    if (g_coapTaskId == NULL) {
        g_terminalFlag = 0;
        SOFTBUS_PRINT("[DISCOVERY] create task fail\n");
        return NSTACKX_EFAILED;
    }
#else
    if (g_coapTaskId != -1) {
        return NSTACKX_EOK;
    }

    ThreadAttr attr = {"coap_listen_task", 0x800, 20, 0, 0};
    int error = CreateThread((Runnable)CoapReadHandle, NULL, &attr, (unsigned
int *)&g_coapTaskId);
    if (error != 0) {
        g_terminalFlag = 0;
        SOFTBUS_PRINT("[DISCOVERY] create task fail\n");
        return NSTACKX_EFAILED;
    }
#endif
    return NSTACKX_EOK;
}
================================================================================
================
#define TIME_MICRO_SEC 10000
static void CoapReadHandle(unsigned int uwParam1, unsigned int uwParam2,
unsigned int uwParam3, unsigned int uwParam4)
{
    (void)uwParam1;
    (void)uwParam2;
    (void)uwParam3;
    (void)uwParam4;
    int ret;
    fd_set readSet;
    int serverFd = GetCoapServerSocket();
    SOFTBUS_PRINT("[DISCOVERY] CoapReadHandle coin select begin\n");
    while (g_terminalFlag) {
        FD_ZERO(&readSet);
        FD_SET(serverFd, &readSet);
        //通过调用select实现了异步通讯
        ret = select(serverFd + 1, &readSet, NULL, NULL, NULL);
        if (ret > 0) {
            if (FD_ISSET(serverFd, &readSet)) {
                //在端口监听到包是通过HandleReadEvent函数来处理，在这个函数中对收到的数据
包进行处理
                HandleReadEvent(serverFd);//参考【三、当发现端（比如手机）发送广播】
            }
        } else {
            SOFTBUS_PRINT("[DISCOVERY]ret:%d,error:%d\n", ret, errno);
        }
    }
    SOFTBUS_PRINT("[DISCOVERY] CoapReadHandle exit\n");
}
```

2.模块增加到g_publishModule

```
//g_publishModule是啥，要参考上面解析过的：1.2.初始化全局g_publishModule和
g_capabilityData
//传入参数有PublishInfo *info，g_publishModule大部分内容继承PublishInfo
//而PublishInfo结构体的内容，是最早调用PublishService()就需要提供的参数
//通俗讲就是将加入软总线的模块名和它的信息，添加到系统g_publishModule中。
PublishModule *AddPublishModule(const char *packageName, const PublishInfo
*info)
{
    //验证参数
    if (packageName == NULL || g_publishModule == NULL || info == NULL) {
        return NULL;
    }
    if (info->dataLen > MAX_SERVICE_DATA_LEN) {
        return NULL;
    }
    if (FindExistModule(packageName, info->publishId) != NULL) {
        return NULL;
    }
    if (FindFreeModule() == NULL) {
        return NULL;
    }

    //
    int ret;
    for (int i = 0; i < MAX_MODULE_COUNT; i++) {  //#define MAX_MODULE_COUNT 3
        if (g_publishModule[i].used == 1) {
            continue;
        }

        if (ParseCapability(info->capability,
&g_publishModule[i].capabilityBitmap)) {
            return NULL;
        }

        g_publishModule[i].used = 1;
        g_publishModule[i].capabilityData = calloc(1, info->dataLen + 1);
        if (g_publishModule[i].capabilityData == NULL) {
            memset_s(&g_publishModule[i], sizeof(g_publishModule[i]), 0,
sizeof(g_publishModule[i]));
            return NULL;
        }
        g_publishModule[i].dataLength = info->dataLen + 1;
        ret = memcpy_s(g_publishModule[i].capabilityData,
                       g_publishModule[i].dataLength,
                       info->capabilityData, info->dataLen);
        if (ret != 0) {
            free(g_publishModule[i].capabilityData);
            g_publishModule[i].capabilityData = NULL;
            memset_s(&g_publishModule[i], sizeof(g_publishModule[i]), 0,
sizeof(g_publishModule[i]));
            return NULL;
        }
        g_publishModule[i].medium = info->medium;
        g_publishModule[i].publishId = info->publishId;
        ret = memcpy_s(g_publishModule[i].package, MAX_PACKAGE_NAME,
packageName, strlen(packageName));
        if (ret != 0) {
            free(g_publishModule[i].capabilityData);
            g_publishModule[i].capabilityData = NULL;
            memset_s(&g_publishModule[i], sizeof(g_publishModule[i]), 0,
sizeof(g_publishModule[i]));
            return NULL;
        }
        return &g_publishModule[i];
    }
    return NULL;
}
```

3.CoapRegisterDefualtService()

```
Z:\harmony110\foundation\communication\softbus_lite\discovery\discovery_service\
source\coap_service.c

int CoapRegisterDefualtService(void)
{
    DeviceInfo *info = GetCommonDeviceInfo(); //info继承全局变量g_deviceInfo结构体
    if (info == NULL) {
        return ERROR_FAIL;
    }

    char serviceData[MAX_DEFAULT_SERVICE_DATA_LEN] = {0};
    if (sprintf_s(serviceData, sizeof(serviceData), "port:%d", info->devicePort)
== -1) {
        return ERROR_FAIL;
    }

    return NSTACKX_RegisterServiceData(serviceData);
}
==============================================================================
Z:\harmony110\foundation\communication\softbus_lite\discovery\coap\source\nstack
x_common.c

int NSTACKX_RegisterServiceData(const char* serviceData)
{
    if (serviceData == NULL) {
        return NSTACKX_EINVAL;
    }

    if (g_nstackInitState != NSTACKX_INIT_STATE_DONE) {
        return NSTACKX_EFAILED;
    }
    unsigned int serviceLen = strlen(serviceData);
    if (serviceLen >= NSTACKX_MAX_SERVICE_DATA_LEN) {
        return NSTACKX_EINVAL;
    }

    if (RegisterServiceData(serviceData, serviceLen + 1) != NSTACKX_EOK) {
        return NSTACKX_EINVAL;
    }
    return NSTACKX_EOK;
}
==============================================================================
Z:\harmony110\foundation\communication\softbus_lite\discovery\coap\source\nstack
x_device.c

int RegisterServiceData(const char* serviceData, int length)
{
    if (serviceData == NULL) {
        return NSTACKX_EINVAL;
    }

    (void)memset_s(g_localDeviceInfo.serviceData,
sizeof(g_localDeviceInfo.serviceData),
        0, sizeof(g_localDeviceInfo.serviceData));
    if (strcpy_s(g_localDeviceInfo.serviceData, NSTACKX_MAX_SERVICE_DATA_LEN,
serviceData) != EOK)  {
        return NSTACKX_EFAILED;
    }

    (void)length;
    return NSTACKX_EOK;
}
==============================================================================
Z:\harmony110\foundation\communication\softbus_lite\discovery\coap\source\nstack
x_device.c

int RegisterServiceData(const char* serviceData, int length)
{
    if (serviceData == NULL) {
        return NSTACKX_EINVAL;
    }

    (void)memset_s(g_localDeviceInfo.serviceData,
sizeof(g_localDeviceInfo.serviceData),
        0, sizeof(g_localDeviceInfo.serviceData));
    if (strcpy_s(g_localDeviceInfo.serviceData, NSTACKX_MAX_SERVICE_DATA_LEN,
serviceData) != EOK)  {
        return NSTACKX_EFAILED;
    }

    (void)length;
    return NSTACKX_EOK;
}
```

总结:

通过模块信息注册发布服务后，系统会产生2个线程，1个监听网络，另一个监听局域网内UDP的发现端请求。

接入网络执行【二、当接入网络，触发WifiEventTrigger()，开启软总线】

监听到UDP广播【三、当发现端（比如手机）发送广播】

二、当接入网络，触发WifiEventTrigger()，开启软总线

```
Z:\harmony110\foundation\communication\softbus_lite\discovery\discovery_service\
source\discovery_service.c
void WifiEventTrigger(unsigned int para)
{
    DeviceInfo *localDev = GetCommonDeviceInfo();
    if (localDev == NULL) {
        return;
    }

    int ret;
    if (para) {
        char wifiIp[MAX_DEV_IP_LEN] = {0};
        CoapGetIp(wifiIp, MAX_DEV_IP_LEN, 0); //参考1.获取本设备ip
        if (strcmp(wifiIp, "0.0.0.0") == 0) {
            SOFTBUS_PRINT("[DISCOVERY] WifiEventTrigger new event interupt.\n");
            return;
        }
        ret = memcpy_s(localDev->deviceIp, sizeof(localDev->deviceIp), wifiIp,
sizeof(wifiIp));
    } else {
        ret = memset_s(localDev->deviceIp, sizeof(localDev->deviceIp), 0,
sizeof(localDev->deviceIp));
    }
    if (ret != ERROR_SUCCESS) {
        return;
    }

    if (BusManager(para) != ERROR_SUCCESS) { //参考2.BusManager()启动软总线
        SOFTBUS_PRINT("[DISCOVERY] WifiEventTrigger StartBusManager(%d) fail\n",
para);
        return;
    }

    if (CoapRegisterDeviceInfo() != ERROR_SUCCESS) {
        SOFTBUS_PRINT("[DISCOVERY] WifiEventTrigger CoapRegisterDeviceInfo
fail\n");
        return;
    }

    if (DoRegistService(COAP) != ERROR_SUCCESS) {
        SOFTBUS_PRINT("[DISCOVERY] WifiEventTrigger DoRegistService fail\n");
        return;
    }
}
```

1.获取本设备ip

```
Z:\harmony110\foundation\communication\softbus_lite\discovery\coap\source\coap_d
iscover.c
//通过CoapGetIp()循环获取本地设备wifi连接后的IP地址，并放入到deviceInfo->deviceIp中。后续
会使用。
void CoapGetIp(char *ip, int length, int finite)
{
    if (ip == NULL || length != NSTACKX_MAX_IP_STRING_LEN) {
        return;
    }

    g_queryIpFlag = 1;
    int count = finite ? GET_IP_TIMES : GET_IP_INFINITE;
    while (g_queryIpFlag) {
        CoapGetWifiIp(ip, length);//获取ip方式根据内核不同，会有多个方式
        if (CheckIpIsValid(ip, strlen(ip)) == 0) {
            break;
        }

        if (count == 0) {
            break;
        }
        count--;
        usleep(TEN_MS); //#define TEN_MS  (10 * 1000)//usleep 10ms，每次获取IP间隔
10ms
    }
    return;
}
```

2.BusManager()启动软总线

```
Z:\harmony110\foundation\communication\softbus_lite\authmanager\source\bus_manag
er.c
int BusManager(unsigned int startFlag)
{
    if (startFlag == 1) {
        return StartBus();
    } else {
        return StopBus();
    }
}
==============================================================================
int StartBus(void)
{
    if (g_busStartFlag == 1) {
        return 0;
    }
    DeviceInfo *info = GetCommonDeviceInfo();
    if (info == NULL) {
        return ERROR_FAIL;
    }

    g_baseLister.onConnectEvent = OnConnectEvent;   //当存在新连接时调用此函数
    g_baseLister.onDataEvent = OnDataEvent;         //当存在新数据时调用此函数
    //StartListener()函数负责为认证模块提供通道完成初始化
    int authPort = StartListener(&g_baseLister, info->deviceIp);  //参考：2.1.启动
侦听
    if (authPort < 0) {
        SOFTBUS_PRINT("[AUTH] StartBus StartListener fail\n");
        return ERROR_FAIL;
    }
    info->devicePort = authPort;
    //StartSession()函数负责初始化业务的session管理
    int sessionPort = StartSession(info->deviceIp); //参考：2.2.启动会话
    if (sessionPort < 0) {
        SOFTBUS_PRINT("[AUTH] StartBus StartSession fail\n");
        StopListener();
        return ERROR_FAIL;
    }

    AuthMngInit(authPort, sessionPort);
    g_busStartFlag = 1;

    SOFTBUS_PRINT("[AUTH] StartBus ok\n");
    return 0;
}
==============================================================================
//回调函数的处理
//trans_service模块的使用者设置的回调函数将在存在新连接、和新数据时被调用
//比如认证模块通过以下函数完成认证动作：OnConnectEvent()函数中完成对新连接的处理，
OnDataEvent()函数中完成对新数据的处理。

int OnConnectEvent(int fd, const char *ip)
    ProcessConnectEvent(fd, ip);
    return 0;
}

int OnDataEvent(int fd)
```

```
    ProcessDataEvent(fd);
    return 0;
}
```

2.1.启动监听,StartListener(),监听与发现端建立连接

StartListener()函数的底层存在对应不同版本平台的适配函数，这印证了鸿蒙OS各部分解耦的模块化设计思想，针对不同的硬件设备，组合成最适合该设备的OS。比如创建线程时采用了统一的static void WaitProcess(void)函数，而其内部封装了不同底层API的适配代码。

```
Z:\harmony110\foundation\communication\softbus_lite\trans_service\source\libdist
bus\auth_conn_manager.c
//StartListener()函数负责为认证模块提供通道完成初始化
//该函数主要是创建了一个WaitProcess 的线程，该线程用于对g_maxFd 所代表的文件描述符利用
select 函数进行监控，若返回值大于0，则调用ProcessAuthData 函数。该函数完成对建立的链接的数据
进行收发，并对收到的数据进行处理的工作，两个处理事件的函数分别是onConnectEvent和
onDataEvent，即前面注册的两个回调函数。其中onConnectEvent 函数主要是为新建立连接的设备创建
AuthConnNode 类型的节点，并将其加入链表中；onDataEvent 函数是对AuthConnNode节点中的数据成
员进行内存分配及对数据进行处理。
#if defined(__LITEOS_M__) || defined(__LITEOS_RISCV__)
int StartListener(BaseListener *callback, const char *ip)   // *callback，回调函数
{
    if (callback == NULL || ip == NULL) {
        return -DBE_BAD_PARAM;
    }

    g_callback = callback;
    //InitListenFd()函数完成监听TCP socket的创建和监听，其中IP地址和端口号由上层调用者指
定。
    int rc = InitListenFd(ip, SESSIONPORT); //
    if (rc != DBE_SUCCESS) {
        return -DBE_BAD_PARAM;
    }

    osThreadAttr_t attr;
    attr.name = "trans_auth_task";
    attr.attr_bits = 0U;
    attr.cb_mem = NULL;
    attr.cb_size = 0U;
    attr.stack_mem = NULL;
    attr.stack_size = LOSCFG_BASE_CORE_TSK_DEFAULT_STACK_SIZE;
    attr.priority = osPriorityNormal5; // LOSCFG_BASE_CORE_TSK_DEFAULT_PRIO ->
cmsis prio
    //osThreadNew()在不同平台上会有不同的实现，在LITEOS_A和Linux平台上， osThreadNew()会
调用兼容POSIX的pthread_create()完成线程的创建
    g_uwTskLoID = osThreadNew((osThreadFunc_t)WaitProcess, NULL, &attr);//监听新连
接和数据WaitProcess(void)
    if (NULL == g_uwTskLoID) {
        SOFTBUS_PRINT("[TRANS] StartListener task create fail\n");
        return -1;
    }

    SOFTBUS_PRINT("[TRANS] StartListener ok\n");
    return GetSockPort(g_listenFd);
}
==============================================================================
static void WaitProcess(void)//监听新连接和数据WaitProcess(void)
```

```
{
    SOFTBUS_PRINT("[TRANS] WaitProcess begin\n");
    fd_set readSet;
    fd_set exceptfds;

    while (1) {//WaitProcess()使用忙等方式，调用select()来监听listenFd和数据g_dataFd的
信息，如果监听到有数据可读，则进入ProcessAuthData来处理。
        FD_ZERO(&readSet);
        FD_ZERO(&exceptfds);
        FD_SET(g_listenFd, &readSet);
        if (g_dataFd >= 0) {
            FD_SET(g_dataFd, &readSet);
            FD_SET(g_dataFd, &exceptfds);
        }
        int ret = select(g_maxFd + 1, &readSet, NULL, &exceptfds, NULL);//监听
g_listenFd和数据g_dataFd的信息
        if (ret > 0) {//如果监听到有数据可读，则进入ProcessAuthData来处理。
            if (!ProcessAuthData(g_listenFd, &readSet)) {//【重要的监听】参考：
                SOFTBUS_PRINT("[TRANS] WaitProcess ProcessAuthData fail\n");
                StopListener();
                break;
            }
        } else if (ret < 0) {//如果发现g_dataFd有异常信息，则将其关闭。其中g_dataFd是由
listenFd监听到连接时创建的socket。
            if (errno == EINTR || (g_dataFd > 0 && FD_ISSET(g_dataFd,
&exceptfds))) {
                SOFTBUS_PRINT("[TRANS] errno == EINTR or g_dataFd is in
exceptfds set.\n");
                CloseAuthSessionFd(g_dataFd);
                continue;
            }
            SOFTBUS_PRINT("[TRANS] WaitProcess select fail, stop listener\n");
            StopListener();
            break;
        }
    }
}
```

2.2.启动会话,StartSession()

```
//StartSession()函数负责初始化业务的session管理
//StartSession 该函数只有一个参数，即const char *ip，也就是一个IP，和StartListener函数中
的IP 是一样的。该函数是为全局变量g_sessionMgr 申请空间及初始化，然后根据所给的参数创建socket
文件描述符并监听，之后通过调用StartSelectLoop 函数创建SelectSessionLoop 的线程，该线程将
socket 文件描述符加入集合，并调用select 函数进行监控，若函数的返回值大于0，则调用ProcessData
函数，该函数有两个分支，若socket 未创建session则为其创建session；若已创建session，则处理其
数据部分
Z:\harmony110\foundation\communication\softbus_lite\authmanager\source\bus_manag
er.c
int StartSession(const char *ip)
{
    int port = CreateTcpSessionMgr(true, ip);
    return port;
}
==============================================================================
Z:\harmony110\foundation\communication\softbus_lite\trans_service\source\libdist
bus\tcp_session_manager.c
```

```
int CreateTcpSessionMgr(bool asServer, const char* localIp)
{
    if (localIp == NULL) {
        return TRANS_FAILED;
    }

    if (InitTcpMgrLock() != 0 || GetTcpMgrLock() != 0) {
        return TRANS_FAILED;
    }

    int ret = InitGSessionMgr();
    if (ReleaseTcpMgrLock() != 0 || ret != 0) {
        FreeSessionMgr();
        return TRANS_FAILED;
    }
    g_sessionMgr->asServer = asServer;//该函数是为全局变量g_sessionMgr 申请空间及初始
化
    int listenFd = OpenTcpServer(localIp, DEFAULT_TRANS_PORT);
    if (listenFd < 0) {
        SOFTBUS_PRINT("[TRANS] CreateTcpSessionMgr OpenTcpServer fail\n");
        FreeSessionMgr();
        return TRANS_FAILED;
    }
    int rc = listen(listenFd, LISTEN_BACKLOG);//然后根据所给的参数创建socket 文件描述
符并监听
    if (rc != 0) {
        SOFTBUS_PRINT("[TRANS] CreateTcpSessionMgr listen fail\n");
        CloseSession(listenFd);
        FreeSessionMgr();
        return TRANS_FAILED;
    }
    g_sessionMgr->listenFd = listenFd;

    signal(SIGPIPE, SIG_IGN);
    if (StartSelectLoop(g_sessionMgr) != 0) {//之后通过调用StartSelectLoop 函数创建
SelectSessionLoop 的线程
        SOFTBUS_PRINT("[TRANS] CreateTcpSessionMgr StartSelectLoop fail\n");
        CloseSession(listenFd);
        FreeSessionMgr();
        return TRANS_FAILED;
    }
    return GetSockPort(listenFd);
}
==============================================================================
Z:\harmony110\foundation\communication\softbus_lite\trans_service\source\libdist
bus\tcp_session_manager.c
#if defined(__LITEOS_M__) || defined(__LITEOS_RISCV__)
int StartSelectLoop(TcpSessionMgr *tsm)
{
    if (tsm == NULL) {
        return TRANS_FAILED;
    }

    if (tsm->isSelectLoopRunning) {
        return 0;
    }

    osThreadId_t sessionLoopTaskId;
    osThreadAttr_t attr;
    attr.name = "trans_session_task";
    attr.attr_bits = 0U;
    attr.cb_mem = NULL;
    attr.cb_size = 0U;
    attr.stack_mem = NULL;
    attr.stack_size = LOSCFG_BASE_CORE_TSK_DEFAULT_STACK_SIZE;
    attr.priority = osPriorityNormal5; // LOSCFG_BASE_CORE_TSK_DEFAULT_PRIO ->
cmsis prio
    //该线程将socket 文件描述符加入集合，并调用select 函数进行监控，
    sessionLoopTaskId = osThreadNew((osThreadFunc_t)SelectSessionLoop, (void
*)tsm, &attr);
    if (NULL == sessionLoopTaskId) {
        SOFTBUS_PRINT("[TRANS] StartSelectLoop TaskCreate fail\n");
        return TRANS_FAILED;
    }

    return 0;
}
==============================================================================
Z:\harmony110\foundation\communication\softbus_lite\trans_service\source\libdist
bus\tcp_session_manager.c
static void SelectSessionLoop(TcpSessionMgr *tsm)
{
    if (tsm == NULL) {
        return;
    }
    SOFTBUS_PRINT("[TRANS] SelectSessionLoop begin\n");
    tsm->isSelectLoopRunning = true;
    while (true) {
        fd_set readfds;
        fd_set exceptfds;
        int maxFd = InitSelectList(tsm, &readfds, &exceptfds);
        if (maxFd < 0) {
            break;
        }

        errno = 0;
        int ret = select(maxFd + 1, &readfds, NULL, &exceptfds, NULL);
        if (ret < 0) {
            SOFTBUS_PRINT("RemoveExceptSessionFd\r\n");
            if (errno == EINTR || RemoveExceptSessionFd(tsm, &exceptfds) == 0) {
                continue;
            }
            SOFTBUS_PRINT("[TRANS] SelectSessionLoop close all Session\n");
            CloseAllSession(tsm);
            break;
        } else if (ret == 0) {
            continue;
        } else {//若函数的返回值大于0，则调用ProcessData 函数，
            ProcessData(tsm, &readfds);
        }
    }
    tsm->isSelectLoopRunning = false;
}
==============================================================================
Z:\harmony110\foundation\communication\softbus_lite\trans_service\source\libdist
bus\tcp_session_manager.c
static void ProcessData(TcpSessionMgr *tsm, fd_set *rfds)
{
    if (tsm == NULL || tsm->listenFd == -1) {
        return;
    }
    //该函数有两个分支，若socket 未创建session则为其创建session；若已创建session，则处理其
数据部分
    if (FD_ISSET(tsm->listenFd, rfds)) {
        ProcessConnection(tsm);
        return;
    }

    ProcessSesssionData(tsm, rfds);
}
```

总结：

当软总线启动后，就会开始监听新连接，OnConnectEvent()函数中完成对新连接的处理, OnDataEvent()函数中完成对新数据的处理。

三、当发现端（比如手机）发送广播

当鸿蒙手机开启锁屏后，会在局域网内发送UDP广播，当开发板（服务端）检测到广播会调用下面的函数。

1.当检测到局域网内UDP广播包,HandleReadEvent()

```
Z:\harmony110\foundation\communication\softbus_lite\discovery\coap\source\coap_d
iscover.c
static void HandleReadEvent(int fd)
{
    int socketFd = fd;
    unsigned char *recvBuffer = calloc(1, COAP_MAX_PDU_SIZE + 1);
    if (recvBuffer == NULL) {
        return;
    }
    ssize_t nRead;
    nRead = CoapSocketRecv(socketFd, recvBuffer, COAP_MAX_PDU_SIZE);//这个函数读取
socket的内容
    if ((nRead == 0) || (nRead < 0 && errno != EAGAIN &&
        errno != EWOULDBLOCK && errno != EINTR)) {
        free(recvBuffer);
        return;
    }

    COAP_Packet decodePacket;
    (void)memset_s(&decodePacket, sizeof(COAP_Packet), 0, sizeof(COAP_Packet));
    decodePacket.protocol = COAP_UDP;
    COAP_SoftBusDecode(&decodePacket, recvBuffer, nRead);//然后调用
COAP_SoftBusDecode来解码
    PostServiceDiscover(&decodePacket);//解码后的报文由PostServiceDiscover来处理，参
考下面解析
```

```
    free(recvBuffer);
}
==============================================================================
void PostServiceDiscover(const COAP_Packet *pkt)
{
    char *remoteUrl = NULL;
    DeviceInfo deviceInfo;

    if (pkt == NULL) {
        return;
    }

    (void)memset_s(&deviceInfo, sizeof(deviceInfo), 0, sizeof(deviceInfo));
    //关于GetServiceDiscoverInfo()这个解包函数暂时不深挖了，TODO
    if (GetServiceDiscoverInfo(pkt->payload.buffer, pkt->payload.len,
&deviceInfo, &remoteUrl) != NSTACKX_EOK) {
        return;
    }

    char wifiIpAddr[NSTACKX_MAX_IP_STRING_LEN];
    (void)memset_s(wifiIpAddr, sizeof(wifiIpAddr), 0, sizeof(wifiIpAddr));
    (void)inet_ntop(AF_INET, &deviceInfo.netChannelInfo.wifiApInfo.ip,
wifiIpAddr, sizeof(wifiIpAddr));
printf(">>>>>%s:%d\nremoteUrl:%s\nwifiIpAddr:%s\n",__FILE__,__LINE__,remoteUrl,w
ifiIpAddr);//添加打印调试
    if (remoteUrl != NULL) {
        CoapResponseService(pkt, remoteUrl, wifiIpAddr);//通过解析到的手机(发现端)的
地址，应答服务
        free(remoteUrl);
    }
}
==============================================================================
static int CoapResponseService(const COAP_Packet *pkt, const char* remoteUrl,
const char* remoteIp)
{
    int ret;
    CoapRequest coapRequest;
    (void)memset_s(&coapRequest, sizeof(coapRequest), 0, sizeof(coapRequest));
    coapRequest.remoteUrl = remoteUrl;
    coapRequest.remoteIp = remoteIp;
    char *payload = PrepareServiceDiscover();
    if (payload == NULL) {
        return NSTACKX_EFAILED;
    }
printf(">>>>>%s:%d\npayload:%s\n",__FILE__,__LINE__,payload);//添加打印调试
    COAP_ReadWriteBuffer sndPktBuff = {0};
    sndPktBuff.readWriteBuf = calloc(1, COAP_MAX_PDU_SIZE);
    if (sndPktBuff.readWriteBuf == NULL) {
        free(payload);
        return NSTACKX_EFAILED;
    }
    sndPktBuff.size = COAP_MAX_PDU_SIZE;
    sndPktBuff.len = 0;

    ret = BuildSendPkt(pkt, remoteIp, payload, &sndPktBuff);//构建要发回的包
    free(payload);
    if (ret != DISCOVERY_ERR_SUCCESS) {
        free(sndPktBuff.readWriteBuf);
        sndPktBuff.readWriteBuf = NULL;
        return ret;
    }
    coapRequest.data = sndPktBuff.readWriteBuf;
    coapRequest.dataLength = sndPktBuff.len;
    ret = CoapSendRequest(&coapRequest);    //发送
    free(sndPktBuff.readWriteBuf);
    sndPktBuff.readWriteBuf = NULL;

    return ret;
}
```

2.当检测到新连接,ProcessAuthData()

```
Z:\harmony110\foundation\communication\softbus_lite\trans_service\source\libdist
bus\auth_conn_manager.c:85
//无论是新连接请求，还是已有连接中有数据到来，均会进入本函数。
static bool ProcessAuthData(int listenFd, const fd_set *readSet)
{
    if (readSet == NULL || g_callback == NULL || g_callback->onConnectEvent ==
NULL ||
        g_callback->onDataEvent == NULL) {
        return false;
    }

    if (FD_ISSET(listenFd, readSet)) {//判断是否是listenFd上存在消息
        struct sockaddr_in addrClient = {0};
        socklen_t addrLen = sizeof(addrClient);
        //如果是，则说明当前存在新的连接，这时调用accept()完成链接创建，新创建的socket的fd被
存储在g_dataFd中，
        g_dataFd = accept(listenFd, (struct sockaddr *)(&addrClient), &addrLen);
        if (g_dataFd < 0) {
            CloseAuthSessionFd(listenFd);
            return false;
        }
        RefreshMaxFd(g_dataFd);
        //同时调用g_callback->onConnectEvent通知认证模块有新的连接事件发生，并将新创建的
fd和client的IP地址告知认证模块。
        //与此同时，创建g_dataFd时候需要刷新g_maxFd，以保证在WaitProcess()中的下一次
select()操作时中，会监听到g_dataFd上的事件。
        if (g_callback->onConnectEvent(g_dataFd, inet_ntoa(addrClient.sin_addr))
!= 0) {    //参考下面解析OnConnectEvent()
            CloseAuthSessionFd(g_dataFd);
        }
    }
    //如果FD_ISSET()判断出g_dataFd上存在消息，则说明已完成握手的连接向本节点发送了数据，
    if (g_dataFd > 0 && FD_ISSET(g_dataFd, readSet)) {  //参考下面解析OnDataEvent()
        g_callback->onDataEvent(g_dataFd);//这时函数回调g_callback->onDataEvent()，
把控制权返回给调用者，以处理接收到的数据。
    }

    return true;
}
==============================================================================
Z:\harmony110\foundation\communication\softbus_lite\authmanager\source\bus_manag
er.c
int OnConnectEvent(int fd, const char *ip)
```

```
{
    ProcessConnectEvent(fd, ip);    //2.1当建立新连接
    return 0;
}
int OnDataEvent(int fd)
{
    ProcessDataEvent(fd);           //2.2当接收到新数据
    return 0;
}
```

2.1当建立新连接,OnConnectEvent()

```
Z:\harmony110\foundation\communication\softbus_lite\authmanager\source\wifi_auth
_manager.c：192行
void ProcessConnectEvent(int fd, const char *ip)
{
    SOFTBUS_PRINT("[AUTH] ProcessConnectEvent fd = %d\n", fd);
    AuthConn *aconn = FindAuthConnByFd(fd); //通过fd查找验证连接

    aconn = calloc(1, sizeof(AuthConn));  //?  系统声明吗?

    int ret = strcpy_s(aconn->deviceIp, sizeof(aconn->deviceIp), ip);//字符串复制函
数

    aconn->fd = fd;

    ret = AddAuthConnToList(aconn);//添加认证连接到列表
}
```

```
Z:\harmony110\foundation\communication\softbus_lite\authmanager\source\wifi_auth
_manager.c:554
这个函数被调用3次??
数据事件进程
void ProcessDataEvent(int fd)
{
    SOFTBUS_PRINT("[AUTH] ProcessDataEvent fd = %d\n", fd);
    AuthConn *conn = FindAuthConnByFd(fd); //通过fd查找验证连接

    if (conn->db.buf == NULL) {
        conn->db.buf = (char *)malloc(DEFAULT_BUF_SIZE);
        if (conn->db.buf == NULL) {
            return;
        }
        (void)memset_s(conn->db.buf, DEFAULT_BUF_SIZE, 0, DEFAULT_BUF_SIZE);
        conn->db.size = DEFAULT_BUF_SIZE;
        conn->db.used = 0;
    }

    DataBuffer *db = &conn->db;
    char *buf = db->buf;
    int used = db->used;
    int size = db->size;

    int rc = AuthConnRecv(fd, buf, used, size - used, 0);
    if (rc == 0) {
        return;
    } else if (rc < 0) {
        CloseConn(conn);
        return;
    }

    used += rc;
    int processed = ProcessPackets(conn, buf, size, used); //参考下面解析1
    if (processed > 0) {
        used -= processed;
        if (used != 0) {
            if (memmove_s(buf, processed, buf, used) != EOK) {
                CloseConn(conn);
                return;
            }
        }
    } else if (processed < 0) {
        CloseConn(conn);
        return;
    }

    db->used = used;
    SOFTBUS_PRINT("[AUTH] ProcessDataEvent ok\n");
}
```

1.\Z:\harmony110\foundation\communication\softbus_lite\authmanager\source\wifi_auth_manager.c：527行

```
static int ProcessPackets(AuthConn *conn, const char *buf, int size, int used)
{
    int processed = 0;
    while (processed + PACKET_HEAD_SIZE < used) {
        Packet *pkt = ParsePacketHead(buf, processed, used - processed, size);//
参考下面解析1

        int len = pkt->dataLen;//将解析完包的大小，赋值给len
        processed += PACKET_HEAD_SIZE;//+24
        OnDataReceived(conn, pkt, buf + processed);//参考下面解析2
        processed += len;
        free(pkt);
        pkt = NULL;
    }
    return processed;
}
```

1.\Z:\harmony110\foundation\communication\softbus_lite\authmanager\source\wifi_auth_manager.c：485行

```
//解析包头，这个函数返回一个packet的结构体
static Packet *ParsePacketHead(const char *buf, int offset, int len, int size)
{
    unsigned int identifier = GetIntFromBuf(buf, offset); //1.从buf获取识别码

    offset += DEFAULT_INT_LEN; //将偏移量+4
    int module = GetIntFromBuf(buf, offset);//2.从buf获取module
    offset += DEFAULT_INT_LEN;//将偏移量+4
    long long seq = 0;
    if (GetLongFromBuf(buf, offset, &seq) != 0) {//3.从buf获取序列号
        return NULL;
    }

    offset += DEFAULT_LONG_LEN;//将偏移量+4
    int flags = GetIntFromBuf(buf, offset);//4.从buf获取标志位

    offset += DEFAULT_INT_LEN;//将偏移量+4
    int dataLen = GetIntFromBuf(buf, offset);//5.从buf获取数据长度

    SOFTBUS_PRINT("[AUTH] ParsePacketHead module=%d, seq=%lld, flags=%d,
datalen=%d\n", module, seq, flags, dataLen);

    Packet *packet = (Packet *)malloc(sizeof(Packet));
    packet->module = module;
    packet->seq = seq;
    packet->flags = flags;
    packet->dataLen = dataLen;

    return packet;
}
```

2.\Z:\harmony110\foundation\communication\softbus_lite\authmanager\source\wifi_auth_manager.c：446行

```
static void OnDataReceived(AuthConn *conn, const Packet *pkt, const char *data)
{
    SOFTBUS_PRINT("[AUTH] OnDataReceived\n"); //数据接收
    if ((pkt->module > MODULE_HICHAIN) && (pkt->module <= MODULE_AUTH_SDK)) {
        //接收数据的认证接口函数，参考解析1
        AuthInterfaceOnDataReceived(conn, pkt->module, pkt->seq, data, pkt-
>dataLen);
        return;
    }

    cJSON *msg = DecryptMessage(pkt->module, data, pkt->dataLen);
    if (msg == NULL) {
        SOFTBUS_PRINT("[AUTH] OnDataReceived DecryptMessage fail\n");
        return;
    }

    OnModuleMessageReceived(conn, pkt->module, pkt->flags, pkt->seq, msg);
    cJSON_Delete(msg);
    msg = NULL;
}
```

```
Z:\harmony110\foundation\communication\softbus_lite\authmanager\source\auth_inte
rface.c
void AuthInterfaceOnDataReceived(const AuthConn *conn, int module, long long
seqId, const char *data, int dataLen)
{
    SOFTBUS_PRINT("[AUTH] AuthInterfaceOnDataReceived begin\n");
    if (AuthSessionMapInit() != 0) { //认证会话初始化
        return;
    }
    AuthSession *auth = AuthGetAuthSessionBySeqId(seqId);//身份验证-通过序列ID获取验
证会话
    if (auth == NULL) {
        auth = AuthGetNewAuthSession(conn, seqId, g_authSessionId);//身份验证-获取
新的验证会话
        if (auth == NULL) {
            return;
        }
        ++g_authSessionId;
    }

    switch (module) {
        case MODULE_AUTH_SDK:
            AuthProcessReceivedData(auth->sessionId, data, dataLen);//身份验证线程
接收数据
            break;
        default:
            break;
    }
    return;
}
```

2.2当接收到新数据,OnDataEvent()

```
Z:\harmony110\foundation\communication\softbus_lite\authmanager\source\wifi_auth
_manager.c
void ProcessDataEvent(int fd)
{
    SOFTBUS_PRINT("[AUTH] ProcessDataEvent fd = %d\n", fd);
    AuthConn *conn = FindAuthConnByFd(fd);
    if (conn == NULL) {
        SOFTBUS_PRINT("ProcessDataEvent get authConn fail\n");
        return;
    }

    if (conn->db.buf == NULL) {
        conn->db.buf = (char *)malloc(DEFAULT_BUF_SIZE);
        if (conn->db.buf == NULL) {
            return;
        }
        (void)memset_s(conn->db.buf, DEFAULT_BUF_SIZE, 0, DEFAULT_BUF_SIZE);
        conn->db.size = DEFAULT_BUF_SIZE;
        conn->db.used = 0;
    }

    DataBuffer *db = &conn->db;
    char *buf = db->buf;
    int used = db->used;
    int size = db->size;
    printf(">>>>>%s:%d\nbuf:%s\n",__FILE__,__LINE__,buf);
    int rc = AuthConnRecv(fd, buf, used, size - used, 0);//认证连接的返回信息
    if (rc == 0) {
        return;
    } else if (rc < 0) {
        CloseConn(conn);
        return;
    }
    used += rc;
    int processed = ProcessPackets(conn, buf, size, used);//对收到的包处理
    if (processed > 0) {
        used -= processed;
        if (used != 0) {
            if (memmove_s(buf, processed, buf, used) != EOK) {
                CloseConn(conn);
                return;
            }
        }
    } else if (processed < 0) {
        CloseConn(conn);
        return;
    }
    db->used = used;
    SOFTBUS_PRINT("[AUTH] ProcessDataEvent ok\n");
}
==============================================================================
Z:\harmony110\foundation\communication\softbus_lite\authmanager\source\auth_conn
.c
int AuthConnRecv(int fd, char *buf, int offset, int count, int timeout)
{
    if ((buf == NULL) || (offset < 0) || (count <= 0) || (offset + count <= 0))
{
        return -1;
    }

    return TcpRecvData(fd, buf + offset, count, timeout);
}
==============================================================================
Z:\harmony110\foundation\communication\softbus_lite\trans_service\source\utils\t
cp_socket.c
int TcpRecvData(int fd, char *buf, int len, int timeout)
{
    return TcpRecvMessages(fd, buf, len, timeout, 0);
}
==============================================================================
Z:\harmony110\foundation\communication\softbus_lite\trans_service\source\utils\t
cp_socket.c
static int32_t TcpRecvMessages(int fd, char *buf, uint32_t len, int timeout, int
flags)
{printf(">>>>>%s:%d:%s()\n",__FILE__,__LINE__,__FUNCTION__);
    if (fd < 0 || buf == NULL || len == 0 || timeout < 0) {
        return -1;
    }
printf(">>>>>%s:%d\nfd:%d,len:%d,flags:%d\nTcpRecvMessages:buf:%s\n",__FILE__,__
LINE__,fd,len,flags,buf);
    errno = 0;
    int32_t rc = recv(fd, buf, len, flags);
printf(">>>>>%s:%d\nfd:%d,len:%d,flags:%d,rc:%d\nTcpRecvMessages:buf:%s\n",__FIL
E__,__LINE__,fd,len,flags,rc,buf);
    if ((rc == -1) && (errno == EAGAIN)) {
        rc = 0;
    } else if (rc <= 0) {
        rc = -1;//rc = -1; //这里rc = -1，程序在这里跳出了，但是却找不到int32_t rc = recv(fd, buf, len, flags);的来源
    }
    return rc;
}
//通过添加2条打印信息发现，经过recv(fd, buf, len, flags)之后，buf有了数据，这数据来自手机端，并加密了。关于recv()打开下面文件
//Z:\harmony110\device\hisilicon\hispark_pegasus\sdk_liteos\third_party\lwip_sack\include\lwip\sockets.h:1589
//在函数的描述里可以看做，这里的recv()是作为API提供的，所以只能挖到这里了。
```

auth.设备认证机制

authmanager【提供设备认证机制和设备知识库管理】

```
当发现有请求时，调用ProcessDataEvent函数，收包，检验包头，根据数据包的类型确定不同的处理方式。类型主要包括以下三种：
MODULE_AUTH_SDK       加密数据类型
MODULE_TRUST_ENGINE  可信类型，直接进行数据传输
MODULE_CONNECTION   进行ip及设备认证

├── BUILD.gn
├── include
|    ├── auth_conn.h
|    ├── auth_interface.h
|    ├── bus_manager.h
|    ├── msg_get_deviceid.h
|    └── wifi_auth_manager.h
└── source
     ├── auth_conn.c【提供发送、接收、认证、获取秘钥功能】
     ├── auth_interface.c【管理各个会话节点、各个链接节点、各个秘钥节点，提供包括增删改查等功能】
     ├── bus_manager.c【主要通过deviceIp创建两个不同的listen，主要用来监听系统上有哪些device及新的device节点的创建;其中有两个回调函数OnConnectEvent和OnDataEvent,分别是用来处理设备节点的基本操作及节点数据的处理】
     ├── msg_get_deviceid.c【提供以cJSON格式获取各个设备的信息，包括设备id、链接信息、设备名、设备类型等】
     └── wifi_auth_manager.c【主要实现了连接管理和数据接收功能。连接管理包括连接的建立、断开及连接的查找。数据接收包括数据获取、包头及包长等的校验，并且为了简化包头数据读取，单独实现了对一个int型和一个long型数据的接收函数】
```

reference:

[communication_dsoftbus: DSoftBus capabilities, including discovery, networking, and transmission | 软总线发现、组网、传输功能实现 (gitee.com)]

https://www.cnblogs.com/hatsune/p/14328148.html